KTH, Dept. Of  Electronics
System (ES), School of ICT

Last Revision:
October 28, 2010

# Laboration 1

# IL2200 ASIC Design

## Introduction to High Level Synthesis

Name :

________________________________________

Personal Number :

________________________________________

Assistant :

________________________________________

Date :

________________________________________

**NOTE: Make it sure that you have downloaded the latest version of lab manual from the course website.**

## 1. Introduction

High level synthesis promises to be one of the solutions to cope with significant increase in the demand for design productivity beyond the start of the art methods. It also offers possibilities to explore the design space in an efficient way by dealing with higher abstraction levels and fast implementation ways to prove the feasibility of algorithms.

In this lab we will try to explore these different possibilities using a HLS tool. We will demonstrate and you will learn about benefits of implementing a design at higher level of abstraction and controlling the generation of RTL using various HLS constraints.

## 2. G.A.U.T.

G.A.U.T. is a HLS (High Level Synthesis) tool developed at the Universite de Bretagne Sud (UB). Lab-STICC laboratory. G.A.U.T. generates RTL descriptions from a pure bit-accurate algorithmic specification described in C/C++ language. G.A.U.T. fits design flows upstream and targets FPGA and ASICs. The basic principle followed by the tool is shown in the following figure.

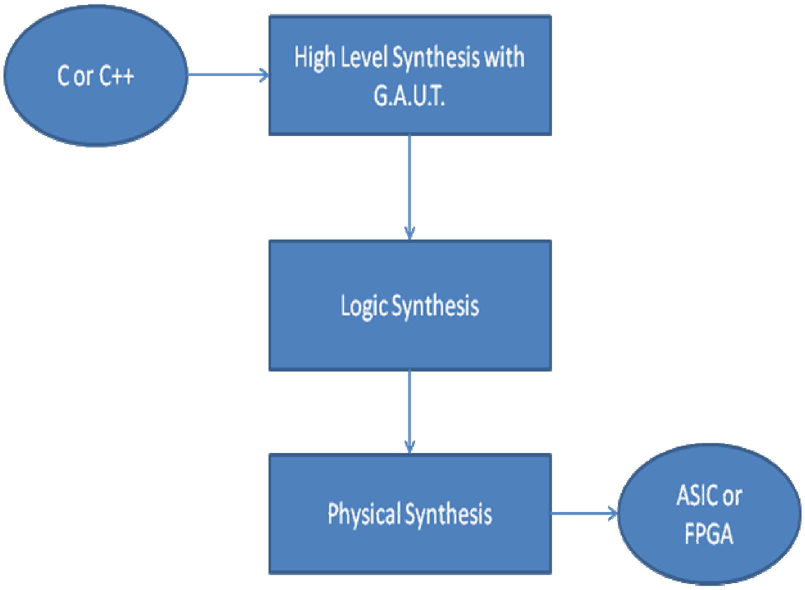


*Figure 1 : G.A.U.T. Design Flow*

Algorithm description is converted into RTL description during the high level synthesis phase and the RTL description is converted in to the logical description suitable for ASIC/FPGA in the next phase. A physical synthesis placing and routing logical gates on a matrix of sites

(ASIC made out of standard cell) or placing and routing CLB on a FPGA (array of CLBs) having a routing topology dynamically reconfigurable.

In this lab we will only be using High Level Synthesis aspect of G.A.U.T. For further details you can consult [1] [2].

## a. G.A.U.T. Interface for HLS

G.A.U.T. is software which has inputs and outputs in the shape of files as well as control options for the HLS. These control options are expressed in the command line or by the means of a graphic interface.

The inputs are:

- a file containing the algorithm to be synthesized: this is a .c or .cpp file
- a library of operators characterized for a given technology target: this is a .lib file.

The outputs are :

- An option file containing VHDL RTL code : this is a .vhd file. It has the same name as the .c, but with a .vhd suffix.
- A textual description of the chronogram (timing diagram) of the I/O of the circuit: the .mem file.
- Other files generated to interface G.A.U.T. with other tools for synthesis.

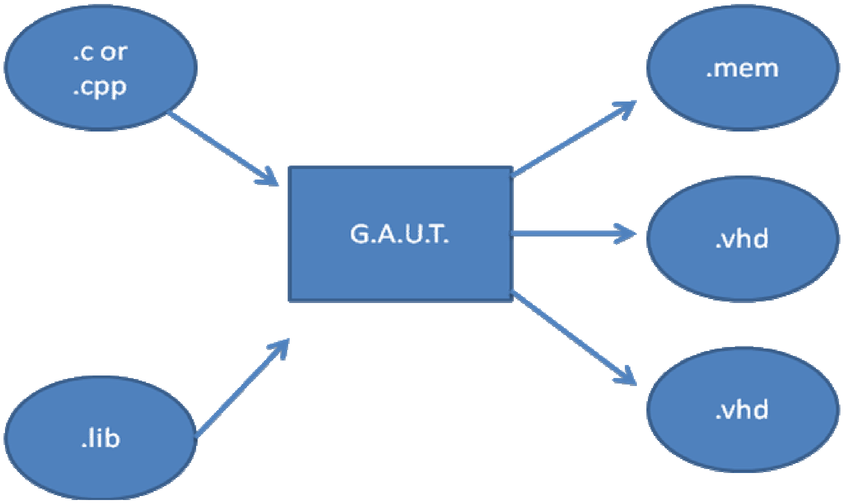


***Figure 2 : G.A.U.T. Interface***

## b. G.A.U.T. HLS Flow

G.A.U.T. takes the functional description of a circuit in form of C code which generates a CDFG. HLS phase takes this CDFG and generates a VHDL RTL after applying scheduling, allocation and binding. Different constraints can be applied on scheduling, allocation and binding phase to explore design space effectively and efficiently. In this lab you will learn to apply these constraints and view the effects of these steps for variations.

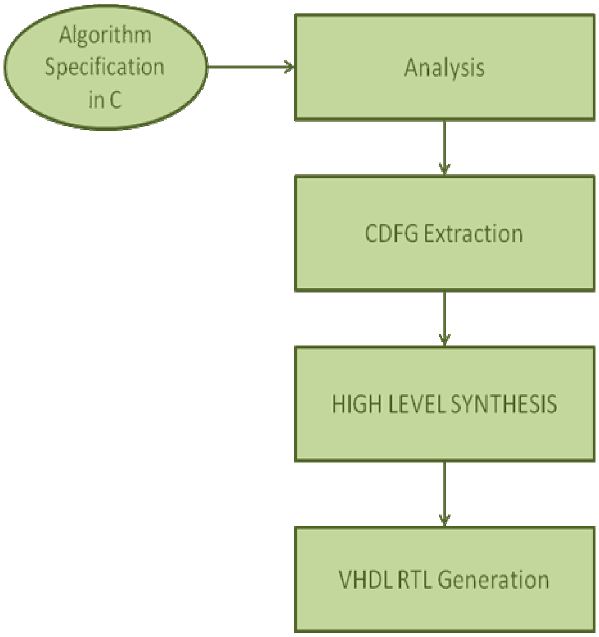


***Figure 3 : G.A.U.T. HLS Flow***

## c. Structure of the synthesized circuit

A circuit synthesized by G.A.U.T. has the following structure:

- A processing unit (PU)
- control signals: clk, reset, enable. When enable = ‘0’, circuit is frozen.
- IO signals connected to external buses with the circuit. “inputs” and “outputs” are the IO as specified in the port of VHDL entity. The external buses do not make parts of the synthesized circuit. **They are there to convey the data between the circuit and external unit storage.** The file .mem describes the scheduling constraints on the data.

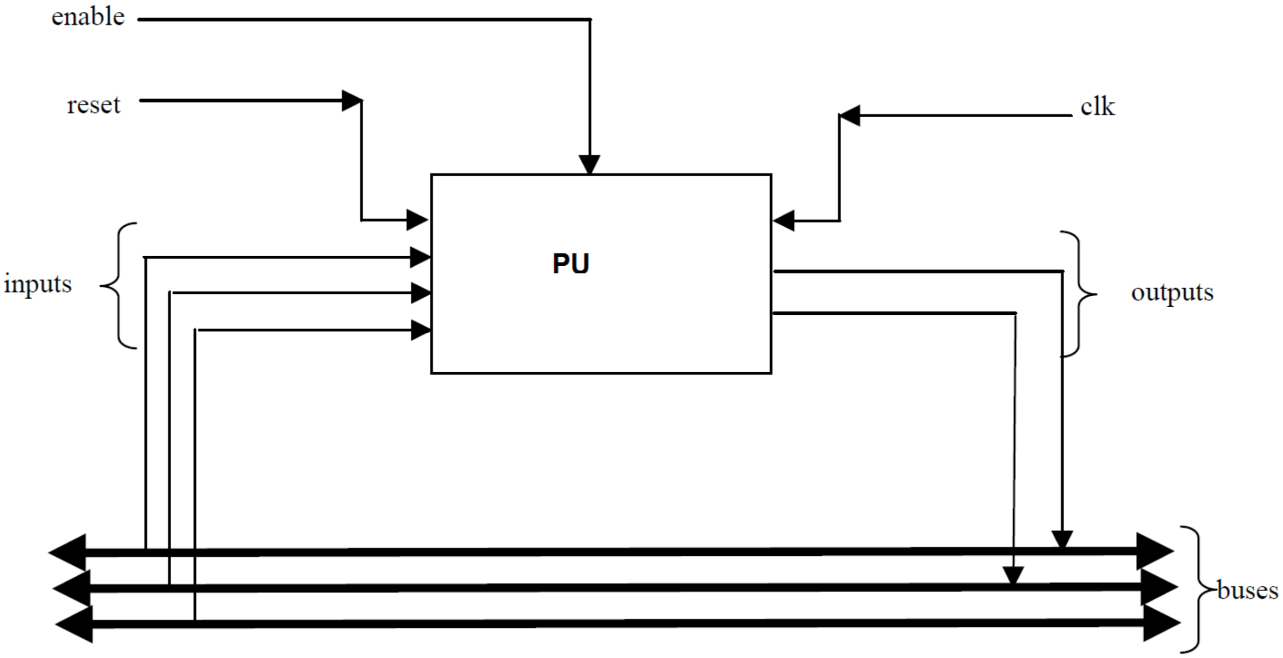


*Figure 4: Synthesized Circuit using G.A.U.T.*

G.A.U.T. can also synthesize the circuit using the Memory unit. It works on exactly the same principle as synthesizing without memory unit. All aging variables and both static and constant variables are stored in the memory, if circuit is synthesized with the memory unit.

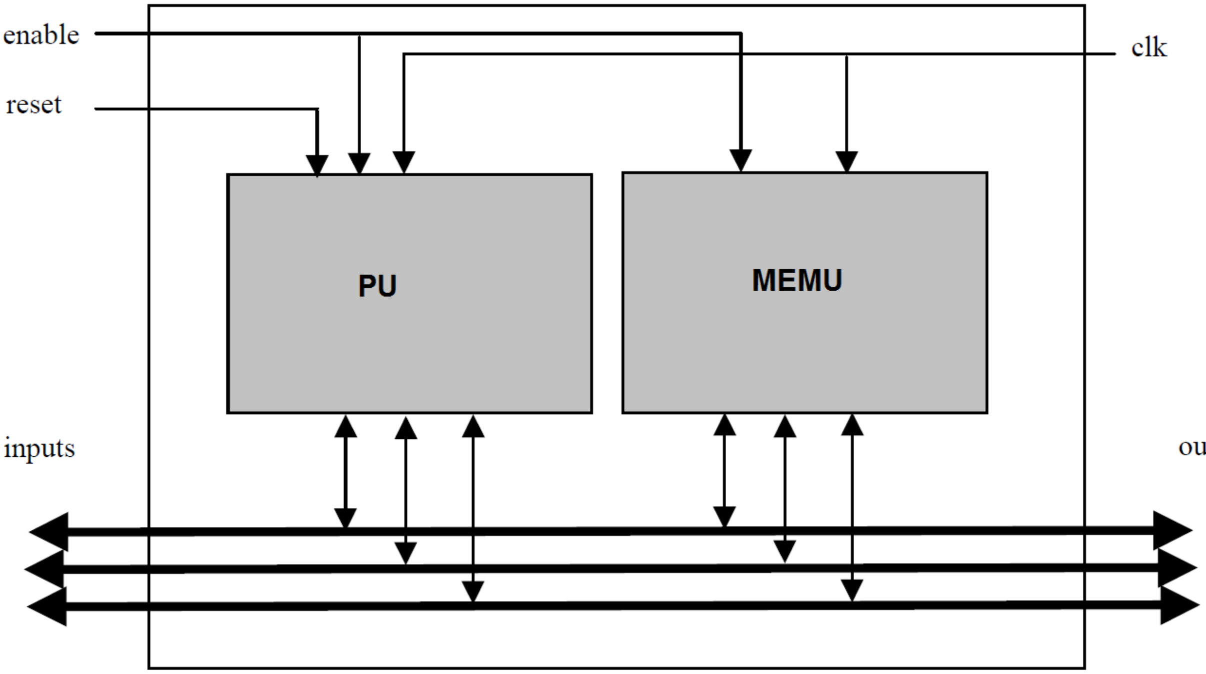


*Figure 5: Synthesized Circuit with MEMU using G.A.U.T.*

## 3. Directory Structure

The directory structure shown in Figure 6 is used for this lab course. Assume that you are working in the lab1/ directory, and then this is your current working directory.

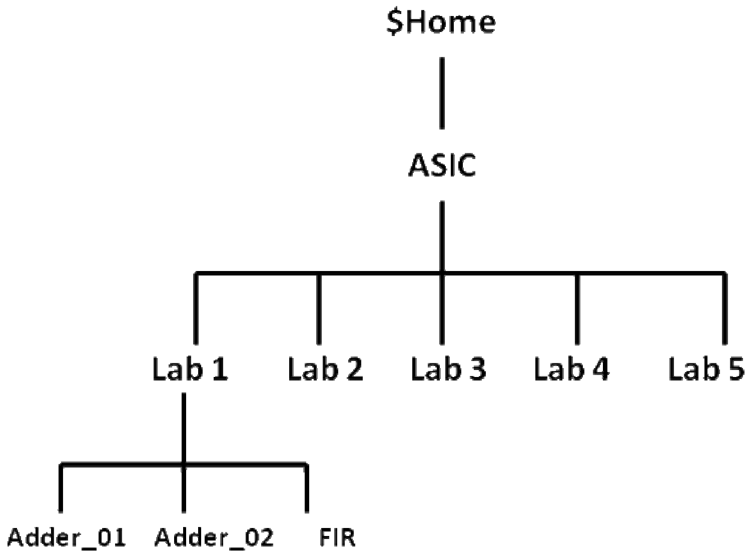


*Figure 6 : Directory Structure*

## 4. An Example using G.A.U.T.

**Objective:**

The primary objective is to give you a quick, hands on tour of the HLS process using G.A.U.T. Upon completion of this exercise you will be able to describe the basic steps involved in the synthesis process and how to explore the design space using various HLS constraints.

**Set of Tasks:**

- If you are working on a windows machine then simply goto :

  **Start -> Programs -> G.A.U.T.**

- If you are working on a linux machine then perform the following steps to start the G.A.U.T. software:
  - Open terminal
  - Make sure that the tool is included in your PATH
  - Type "Gaut &"
  - Another option if you did not set the PATH "cd /afs/ict.kth.se/gaut/2.4.3/" and then "./Gaut &"

Figure 7 shows the main interface of the tool with eight colored boxes. Every box has a specific purpose and will be described briefly later.

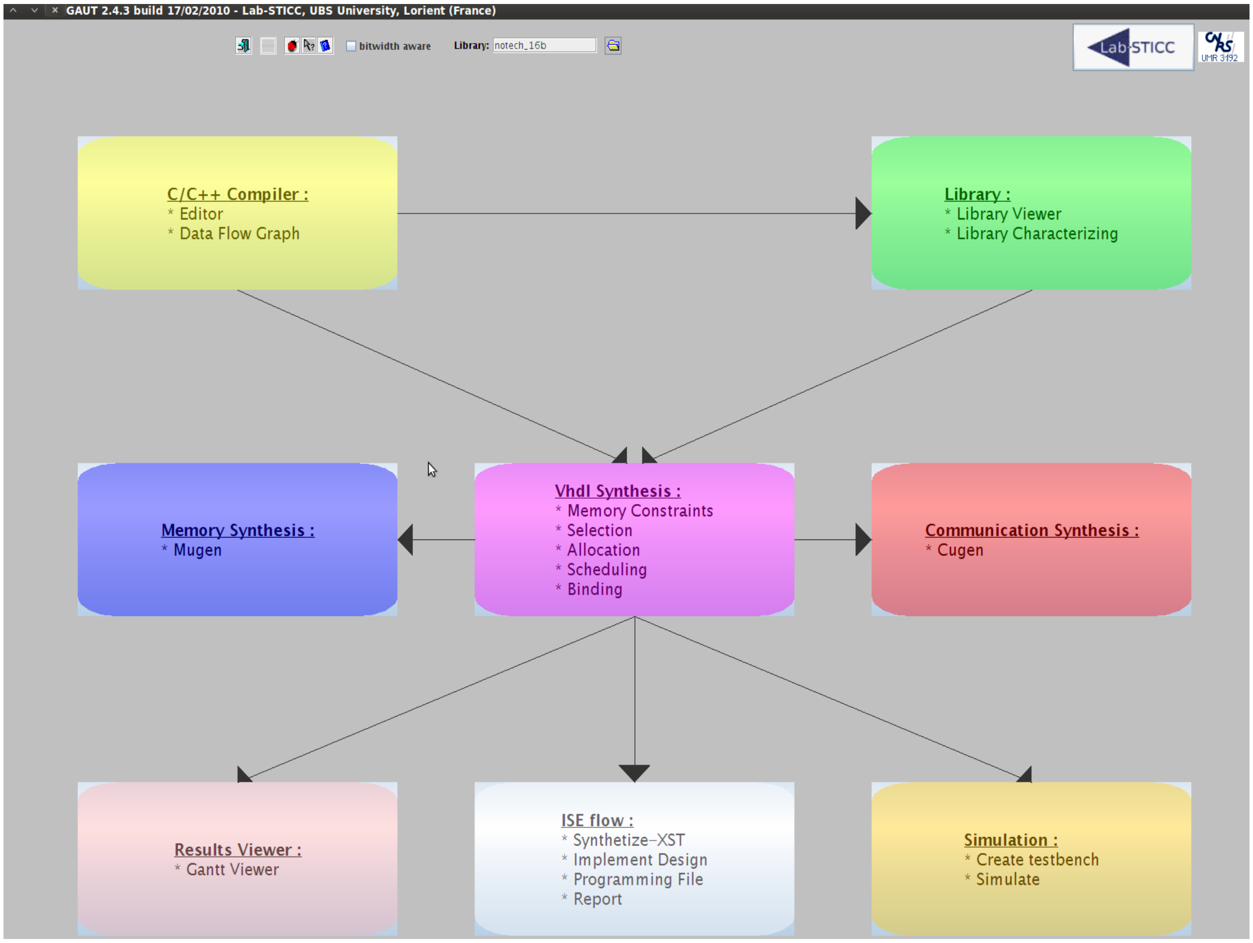
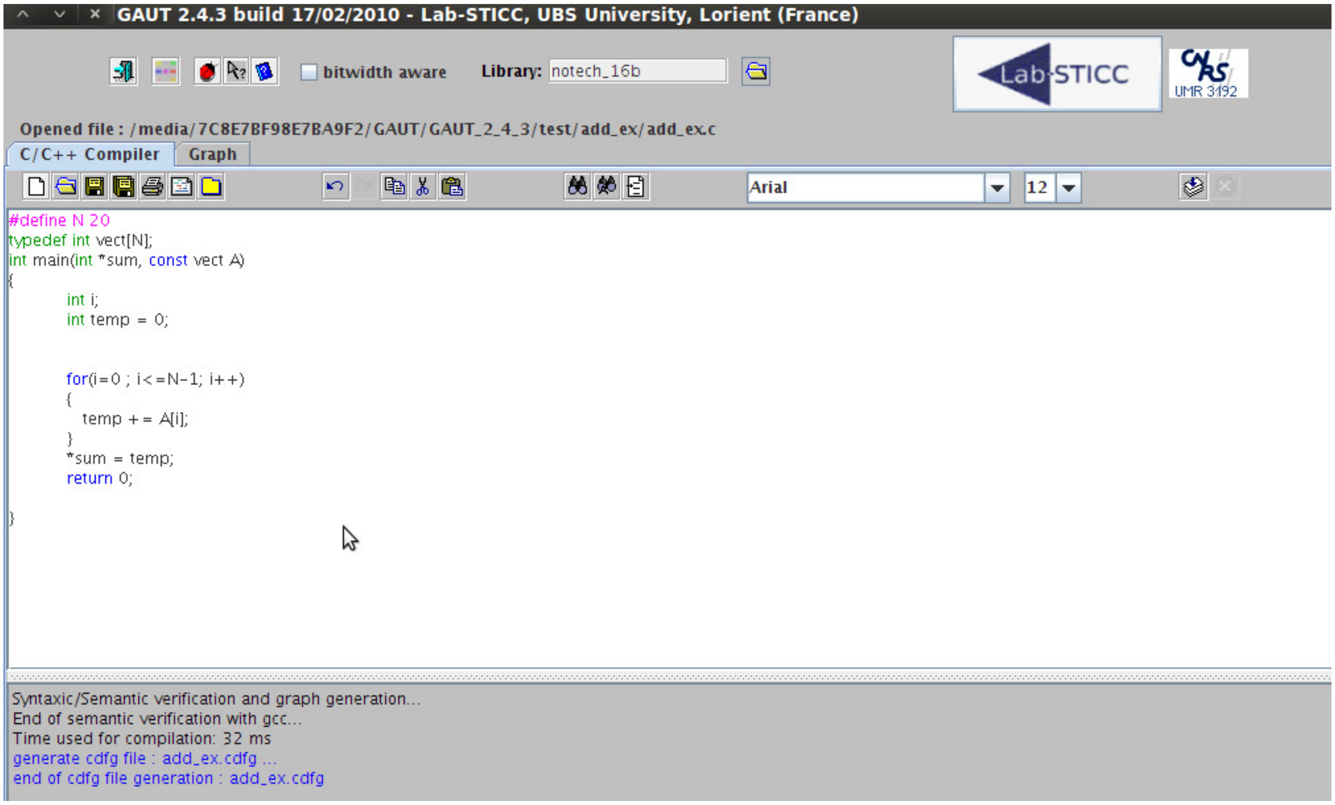


*Figure 7 : G.A.U.T. Main Window*

## a. STEP1 : Compiling the C Code

We will start by compiling a C code using G.A.U.T. This is a simple C code which adds 20 numbers and store the result in the variable "sum".

- **Click on the Yellow box stating "C/C++ Compiler".** (This phase makes it sure that the algorithm specified in C is correct)
- **Load the "adder_1.c" file by clicking on the "open" icon and following the Path :** "$HOME/ASIC/Lab1/adder_1/adder_1.c"
- **Compile the code by clicking on the "compile button"** .


GAUT 2.4.3 build 17/02/2010 - Lab-STICC, UBS University, Lorient (France)
Lab-STICC
CNRS UMR 3192
C/C++ Compiler
Graph
Arial
12

□ bitwidth aware

Library: notech_16b

Opened file : /media/7C8E7BF98E7BA9F2/GAUT/GAUT_2_4_3/test/add_ex/add_ex.c

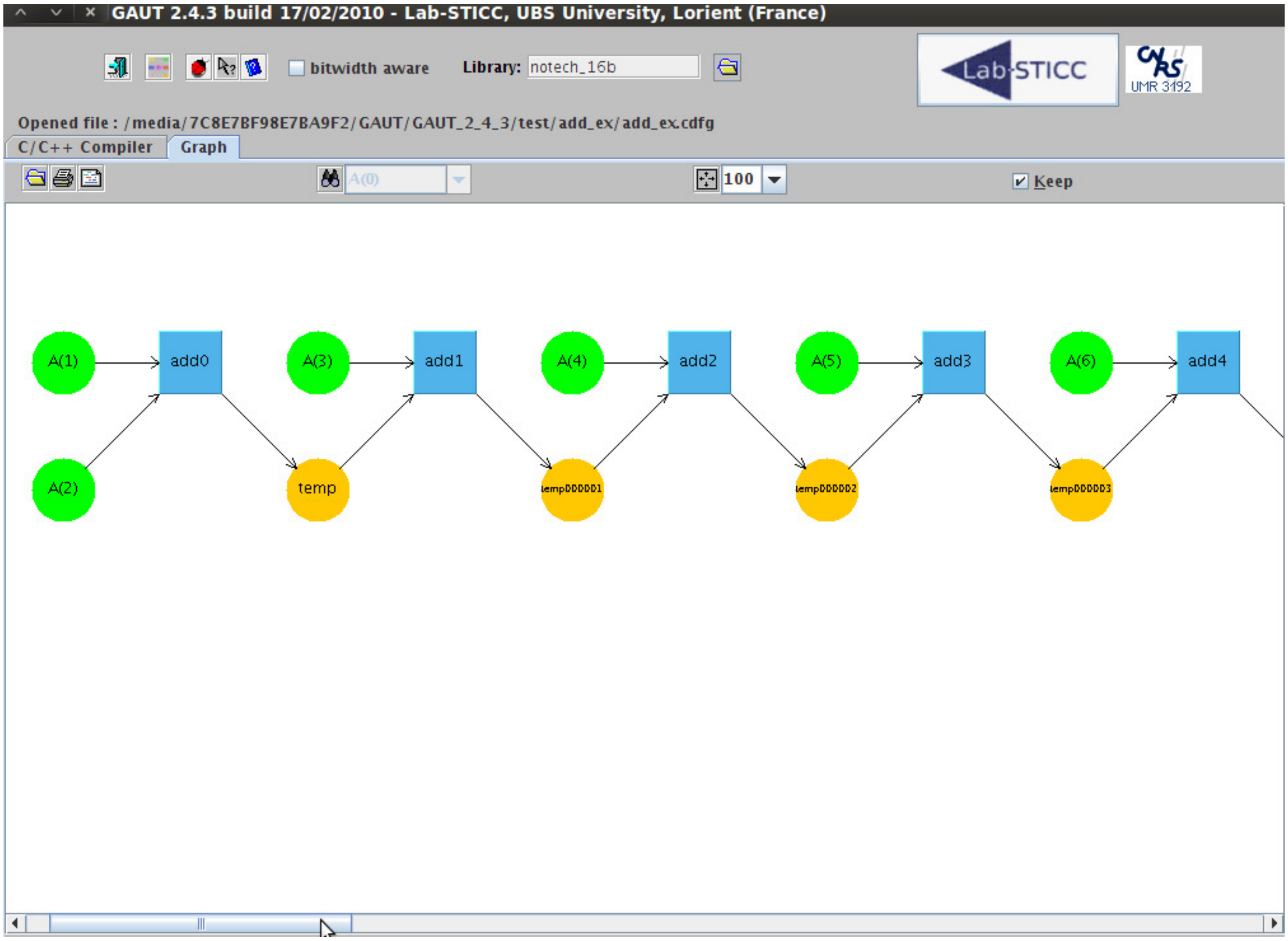
```
#define N 20
typedef int vect[N];
int main(int *sum, const vect A)
{
        int i;
        int temp = 0;


        for(i=0 ; i<=N-1; i++)
        {
          temp += A[i];
        }
        *sum = temp;
        return 0;

}
```

```
Syntaxic/Semantic verification and graph generation...
End of semantic verification with gcc...
Time used for compilation: 32 ms
generate cdfg file : add_ex.cdfg ...
end of cdfg file generation : add_ex.cdfg
```

*Figure 8 : C Editor/Compiler Window*

- **Click on the graph tab and look at the graph by loading "adder_1.cdfg".**
  This is the cdfg corresponding to the design presented in C code. The cdfg contains 19 additions, data values stored in variables A[0] … A[19] and the variables temp, temp 00001 … temp 00017. These variables came from loop unrolling of the code.

  It should also be noted that the technological target library "notech_16b" is a factitious library. Different sets of technological libraries can be characterized by G.A.U.T. and are beyond the scope of this lab. Interested readers can consult [1] for further details.



GAUT 2.4.3 build 17/02/2010 - Lab-STICC, UBS University, Lorient (France)

☐ bitwidth aware

Library: notech_16b

Opened file : /media/7C8E7BF98E7BA9F2/GAUT/GAUT_2_4_3/test/add_ex/add_ex.cdfg

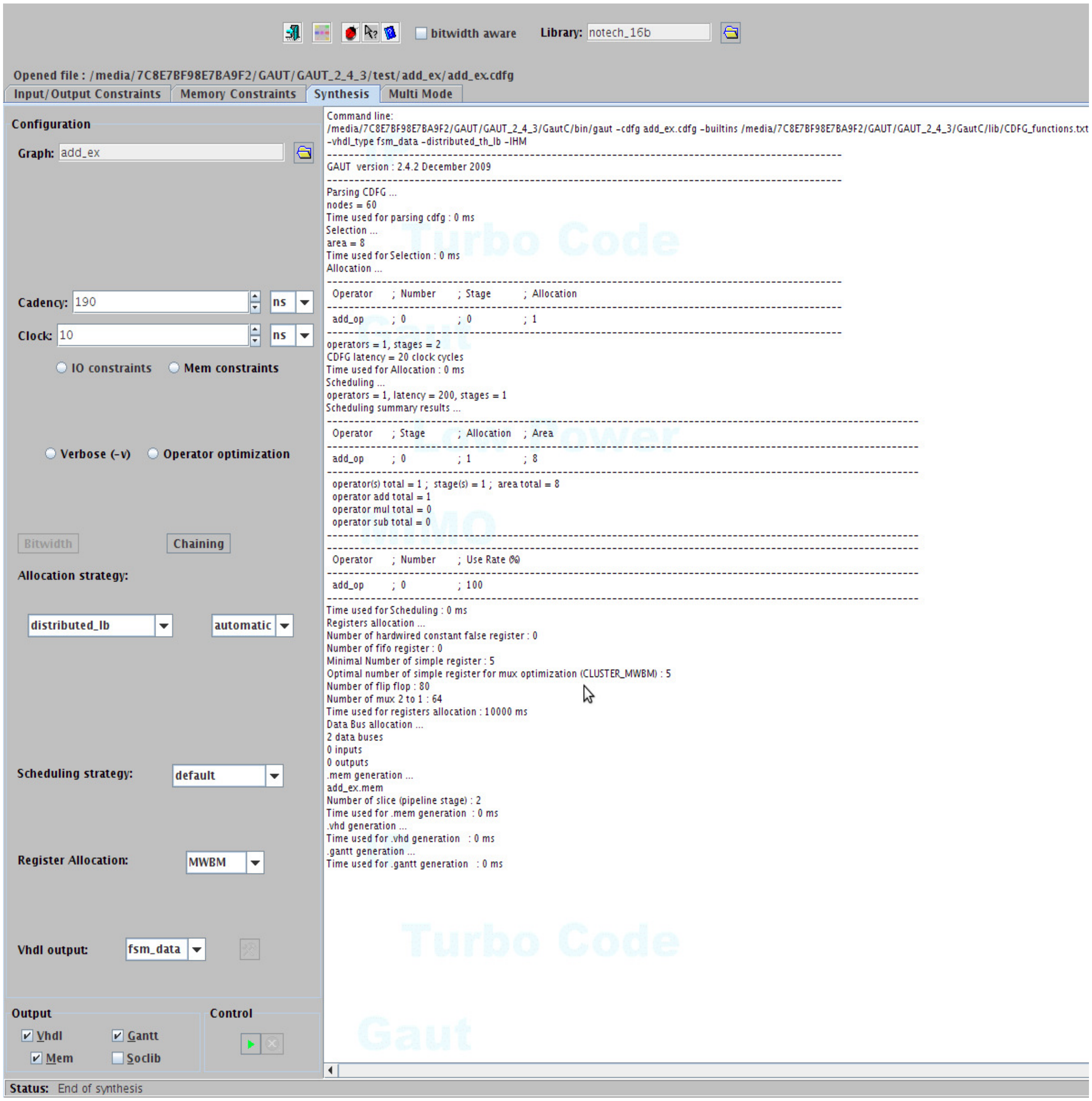
*Figure 9 : CDFG of the C code*

- **Click on the button and you will come back to the main window as shown in the Figure 7.**

## b. STEP 2 : Synthesis procedure

We will apply different constraints on this design for the synthesis purpose. One can use these constraints the change the scheduling, allocation and binding parameters in multiple ways.

- **Click on the Purple box "VHDL synthesis".**



Opened file : /media/7C8E7BF98E7BA9F2/GAUT/GAUT_2_4_3/test/add_ex/add_ex.cdfg

Graph: add_ex

Cadency: 190 ns

Clock: 10 ns

Allocation strategy: distributed_lb automatic

Scheduling strategy: default

Register Allocation: MWBM

Vhdl output: fsm_data

```
Command line:
/media/7C8E7BF98E7BA9F2/GAUT/GAUT_2_4_3/GautC/bin/gaut -cdfg add_ex.cdfg -builtins /media/7C8E7BF98E7BA9F2/GAUT/GAUT_2_4_3/GautC/lib/CDFG_functions.txt
-vhdl_type fsm_data -distributed_th_lb -IHM
--------------------------------------------------------------------------------
GAUT  version : 2.4.2 December 2009
--------------------------------------------------------------------------------
Parsing CDFG ...
nodes = 60
Time used for parsing cdfg : 0 ms
Selection ...
area = 8
Time used for Selection : 0 ms
Allocation ...
--------------------------------------------------------------------------------
 Operator     ;  Number      ;  Stage         ;  Allocation
--------------------------------------------------------------------------------
 add_op       ;  0           ;  0             ;  1
--------------------------------------------------------------------------------
operators = 1, stages = 2
CDFG latency = 20 clock cycles
Time used for Allocation : 0 ms
Scheduling ...
operators = 1, latency = 200, stages = 1
Scheduling summary results ...
----------------------------------------------------------------------------------------------
 Operator     ;  Stage       ;  Allocation   ;  Area
----------------------------------------------------------------------------------------------
 add_op       ;  0           ;  1            ;  8
----------------------------------------------------------------------------------------------
 operator(s) total = 1 ;  stage(s) = 1 ;  area total = 8
 operator add total = 1
 operator mul total = 0
 operator sub total = 0
----------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------
 Operator     ;  Number      ;  Use Rate (%)
----------------------------------------------------------------------------------------------
 add_op       ;  0           ;  100
----------------------------------------------------------------------------------------------
Time used for Scheduling : 0 ms
Registers allocation ...
Number of hardwired constant false register : 0
Number of fifo register : 0
Minimal Number of simple register : 5
Optimal number of simple register for mux optimization (CLUSTER_MWBM) : 5
Number of flip flop : 80
Number of mux 2 to 1 : 64
Time used for registers allocation : 10000 ms
Data Bus allocation ...
2 data buses
0 inputs
0 outputs
.mem generation ...
add_ex.mem
Number of slice (pipeline stage) : 2
Time used for .mem generation  : 0 ms
.vhd generation ...
Time used for .vhd generation   : 0 ms
.gantt generation ...
Time used for .gantt generation   : 0 ms
```

Status: End of synthesis

*Figure 10 : Synthesis options*

Figure 10 shows the main windows for performing synthesis. As described before, there are various options available. We will discuss few of them for this exercise and rest will be explained in later sections.

**Graph** : The synthesis part takes the CDFG as the input to start performing the synthesis. This is the same graph which was generated in the previous step.

**Cadency** : This is the rate of arrival of the sets of data inputs (sampling rate, iteration interval). In other words, this is the throughput of the design. Cadency is one of the primary constraints for synthesizing the design using G.A.U.T.

**Clock** : You can use this field to specify the desired clock period of the RTL to be generated.

**VHDL output** : The tool can generate different styles of VHDL code. It should be noted that each style of VHDL performs the same type of functionality.

For this exercise set the following fields :

- **Set the cadency value as 190 ns.**
- **Set the clock period as 10 ns.**
- **Set the VHDL output type as "fsm_regs". Keep the rest of the values as default**
- **Synthesize the design by pressing the "control" button.** ▶

Some of the information presented in the report is as follows:

The CDFG parsing step

```
Parsing CDFG . . . nodes = 60
```

The Allocation step

```
Allocation … Operators = 1, stages = 2
```

```
CDFG Latency = 20 clock cycles
```

The scheduling step

```
Scheduling … Operators = 1, Latency = 200 , stages = 1
```

```
Register allocation
```

```
Bus Allocation . . .  2 data buses
```

Question 01 : Can you view the synthesis report generated by G.A.U.T.? Is this is a serial or parallel solution?

________________________________________________

________________________________________________

Question 02 : What is the latency of this design? How may clock steps were taken by the tool to complete the job?

________________________________________________

________________________________________________

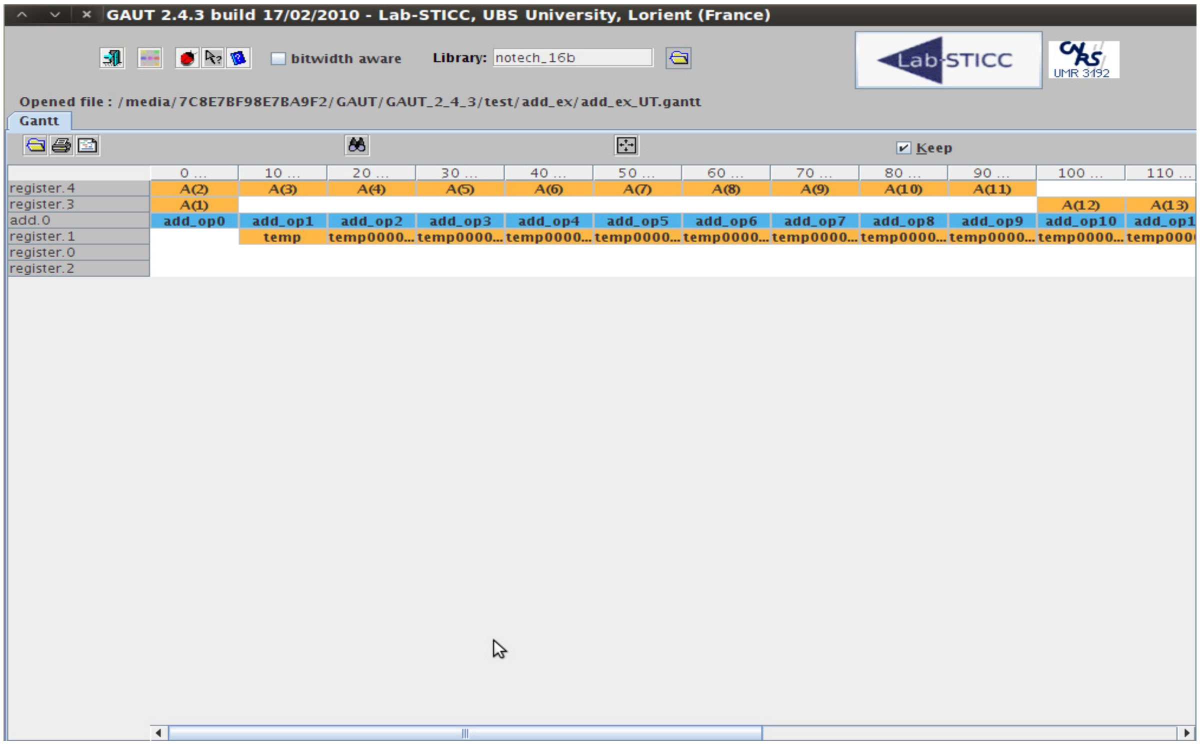
Question 03 : How many adders are used by the tool for generating this solution? What is the minimum delay of a single adder? (You may need to check the Library Viewer)

- **Click on the button and you will come back to the main window as shown in the Figure 8.**

## c. STEP 3 : Viewing the Results

Pink box "Results Viewer", is use to generate Gantt charts of the scheduled operations. This chart explains the results of the scheduling steps. It also contains information about the contents of the circuit in terms of operators and registers.

- **Click on the "Results Viewer" and open the file "adder_1_UT*.gantt".** Horizontally the clue color defines the execution of the operations and the orange color defines the variables and registers in which they are stored. Vertically the names of the operators and registers are defined. (*UT has the same as PU)



GAUT 2.4.3 build 17/02/2010 - Lab-STICC, UBS University, Lorient (France)

□ bitwidth aware Library: notech_16b

Opened file : /media/7C8E7BF98E7BA9F2/GAUT/GAUT_2_4_3/test/add_ex/add_ex_UT.gantt

Gantt

☑ Keep

| | 0 ... | 10 ... | 20 ... | 30 ... | 40 ... | 50 ... | 60 ... | 70 ... | 80 ... | 90 ... | 100 ... | 110 ... |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| register.4 | A(2) | A(3) | A(4) | A(5) | A(6) | A(7) | A(8) | A(9) | A(10) | A(11) | | |
| register.3 | A(1) | | | | | | | | | | A(12) | A(13) |
| add.0 | add_op0 | add_op1 | add_op2 | add_op3 | add_op4 | add_op5 | add_op6 | add_op7 | add_op8 | add_op9 | add_op10 | add_op1 |
| register.1 | | temp | temp0000... | temp0000... | temp0000... | temp0000... | temp0000... | temp0000... | temp0000... | temp0000... | temp0000... | temp000 |
| register.0 | | | | | | | | | | | | |
| register.2 | | | | | | | | | | | | |

*Figure 11 : GANTT Chat of the synthesized design*

Question 04 : How can you co relate your synthesized design with this gantt chart ?

______________________________________________________________

______________________________________________________________

- **Open the following file :**

  "$HOME/ASIC/Lab1/adder_1/adder_1.mem".

  This file shows the I/O chronogram and the temporal access conflicts. The contents of this file tells us :

  -10ns to 0ns : A1 is presented to data bus 1

  -10ns to 0ns : A2 is presented to data bus 2

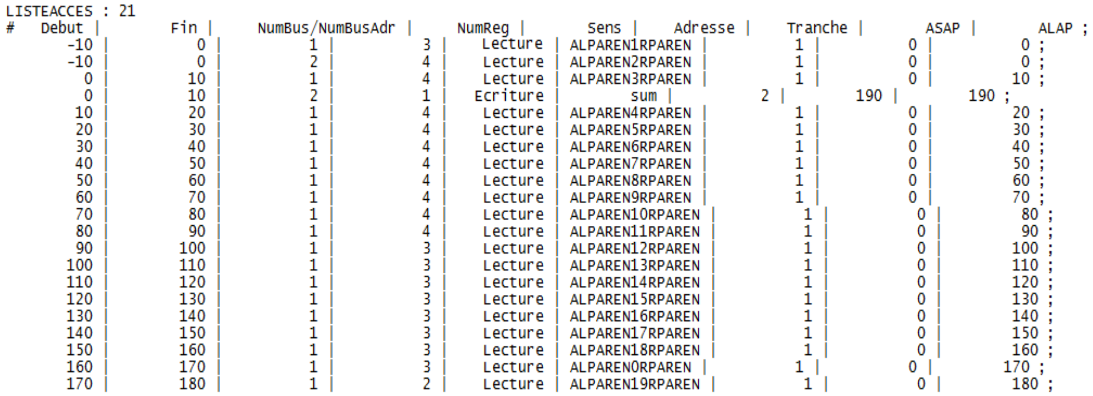
  0ns to 10ns : A3 is presented on data bus 1

  .

  .

  .

  "lecture" means "Read" and "Ecriture" means "Write". This set of information can be used resolving temporal conflicts when synthesizing using memory.



```
LISTEACCES : 21
#   Debut |       Fin |     NumBus/NumBusAdr |     NumReg |       Sens |    Adresse |     Tranche |        ASAP |       ALAP ;
      -10 |         0 |                    1 |          3 |    Lecture | ALPAREN1RPAREN |         1 |          0 |          0 ;
      -10 |         0 |                    2 |          4 |    Lecture | ALPAREN2RPAREN |         1 |          0 |          0 ;
        0 |        10 |                    1 |          4 |    Lecture | ALPAREN3RPAREN |         1 |          0 |         10 ;
        0 |        10 |                    2 |          1 |   Ecriture |        sum |        2 |        190 |        190 ;
       10 |        20 |                    1 |          4 |    Lecture | ALPAREN4RPAREN |         1 |          0 |         20 ;
       20 |        30 |                    1 |          4 |    Lecture | ALPAREN5RPAREN |         1 |          0 |         30 ;
       30 |        40 |                    1 |          4 |    Lecture | ALPAREN6RPAREN |         1 |          0 |         40 ;
       40 |        50 |                    1 |          4 |    Lecture | ALPAREN7RPAREN |         1 |          0 |         50 ;
       50 |        60 |                    1 |          4 |    Lecture | ALPAREN8RPAREN |         1 |          0 |         60 ;
       60 |        70 |                    1 |          4 |    Lecture | ALPAREN9RPAREN |         1 |          0 |         70 ;
       70 |        80 |                    1 |          4 |    Lecture | ALPAREN10RPAREN |         1 |          0 |         80 ;
       80 |        90 |                    1 |          4 |    Lecture | ALPAREN11RPAREN |         1 |          0 |         90 ;
       90 |       100 |                    1 |          3 |    Lecture | ALPAREN12RPAREN |         1 |          0 |        100 ;
      100 |       110 |                    1 |          3 |    Lecture | ALPAREN13RPAREN |         1 |          0 |        110 ;
      110 |       120 |                    1 |          3 |    Lecture | ALPAREN14RPAREN |         1 |          0 |        120 ;
      120 |       130 |                    1 |          3 |    Lecture | ALPAREN15RPAREN |         1 |          0 |        130 ;
      130 |       140 |                    1 |          3 |    Lecture | ALPAREN16RPAREN |         1 |          0 |        140 ;
      140 |       150 |                    1 |          3 |    Lecture | ALPAREN17RPAREN |         1 |          0 |        150 ;
      150 |       160 |                    1 |          3 |    Lecture | ALPAREN18RPAREN |         1 |          0 |        160 ;
      160 |       170 |                    1 |          3 |    Lecture | ALPAREN0RPAREN |         1 |          0 |        170 ;
      170 |       180 |                    1 |          2 |    Lecture | ALPAREN19RPAREN |         1 |          0 |        180 ;
```

## d. STEP 4: Simulating the Design*

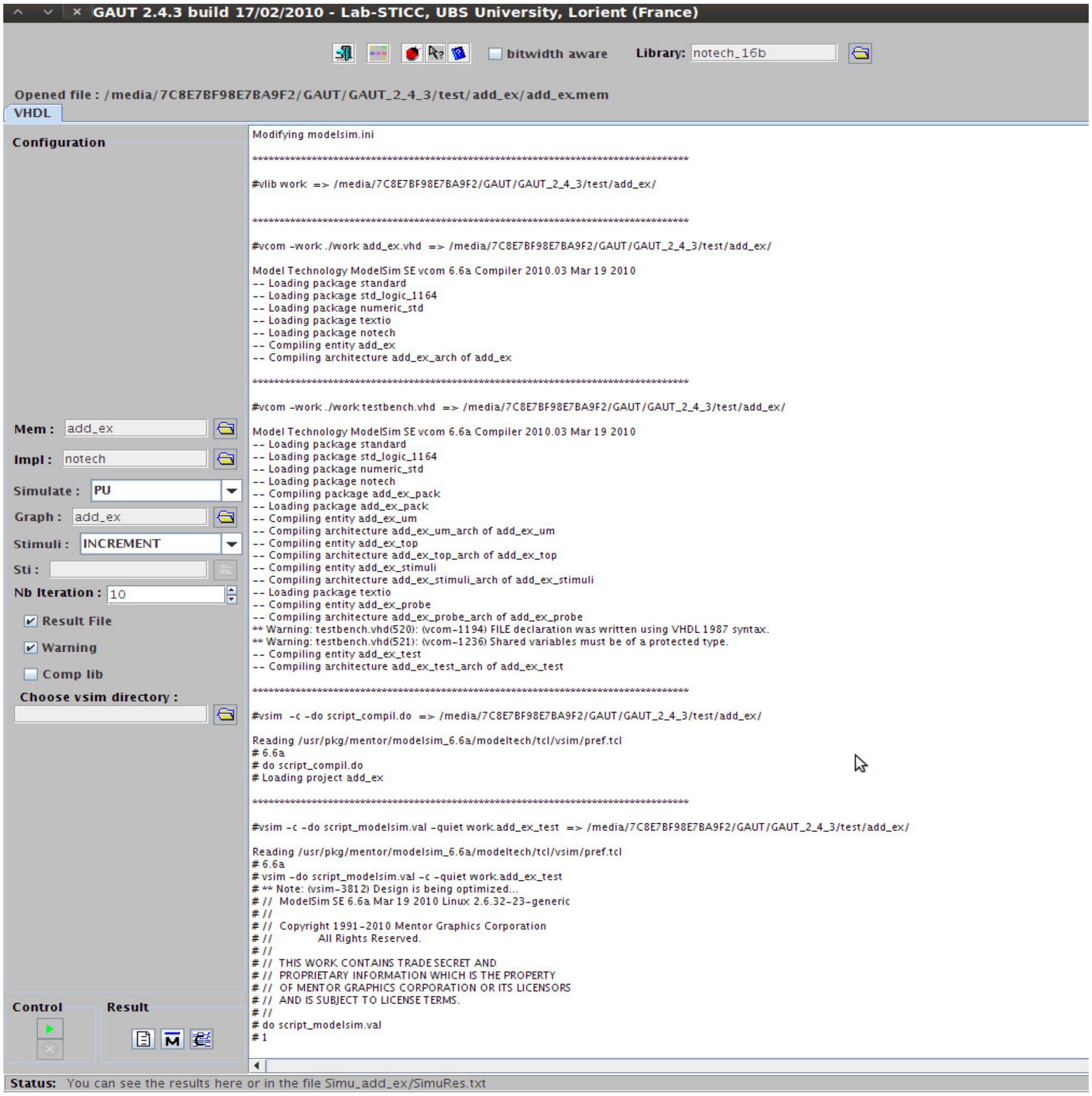
The tool also generates a VHDL RTL of the synthesized design in the working folder.

- **Open the VHDL file which is located at "ASIC\Lab1\adder_1\adder_1.vhd".**

Question 05 : Can you correlate your VHDL with the synthesis report and gantt charts? Are the number of states are matching the number of clock steps ?

______________________________________________________________

- **Click the orange button to simulate the design using Modelsim.**
- **Keep the default values and simulate the design by clicking on the "control” button.**



GAUT 2.4.3 build 17/02/2010 - Lab-STICC, UBS University, Lorient (France)

□ bitwidth aware  Library: notech_16b

Opened file : /media/7C8E7BF98E7BA9F2/GAUT/GAUT_2_4_3/test/add_ex/add_ex.mem

Configuration

Mem : add_ex

Impl : notech

Simulate : PU

Graph : add_ex

Stimuli : INCREMENT

Sti :

Nb Iteration : 10

☑ Result File

☑ Warning

□ Comp lib

Choose vsim directory :

```
Modifying modelsim.ini

*******************************************************************************

#vlib work  => /media/7C8E7BF98E7BA9F2/GAUT/GAUT_2_4_3/test/add_ex/


*******************************************************************************

#vcom -work ./work add_ex.vhd  => /media/7C8E7BF98E7BA9F2/GAUT/GAUT_2_4_3/test/add_ex/

Model Technology ModelSim SE vcom 6.6a Compiler 2010.03 Mar 19 2010
-- Loading package standard
-- Loading package std_logic_1164
-- Loading package numeric_std
-- Loading package textio
-- Loading package notech
-- Compiling entity add_ex
-- Compiling architecture add_ex_arch of add_ex

*******************************************************************************

#vcom -work ./work testbench.vhd  => /media/7C8E7BF98E7BA9F2/GAUT/GAUT_2_4_3/test/add_ex/

Model Technology ModelSim SE vcom 6.6a Compiler 2010.03 Mar 19 2010
-- Loading package standard
-- Loading package std_logic_1164
-- Loading package numeric_std
-- Loading package notech
-- Compiling package add_ex_pack
-- Loading package add_ex_pack
-- Compiling entity add_ex_um
-- Compiling architecture add_ex_um_arch of add_ex_um
-- Compiling entity add_ex_top
-- Compiling architecture add_ex_top_arch of add_ex_top
-- Compiling entity add_ex_stimuli
-- Compiling architecture add_ex_stimuli_arch of add_ex_stimuli
-- Loading package textio
-- Compiling entity add_ex_probe
-- Compiling architecture add_ex_probe_arch of add_ex_probe
** Warning: testbench.vhd(520): (vcom-1194) FILE declaration was written using VHDL 1987 syntax.
** Warning: testbench.vhd(521): (vcom-1236) Shared variables must be of a protected type.
-- Compiling entity add_ex_test
-- Compiling architecture add_ex_test_arch of add_ex_test

*******************************************************************************

#vsim  -c -do script_compil.do  => /media/7C8E7BF98E7BA9F2/GAUT/GAUT_2_4_3/test/add_ex/

Reading /usr/pkg/mentor/modelsim_6.6a/modeltech/tcl/vsim/pref.tcl
# 6.6a
# do script_compil.do
# Loading project add_ex

*******************************************************************************

#vsim -c -do script_modelsim.val -quiet work.add_ex_test  => /media/7C8E7BF98E7BA9F2/GAUT/GAUT_2_4_3/test/add_ex/

Reading /usr/pkg/mentor/modelsim_6.6a/modeltech/tcl/vsim/pref.tcl
# 6.6a
# vsim -do script_modelsim.val -c -quiet work.add_ex_test
# ** Note: (vsim-3812) Design is being optimized...
# //  ModelSim SE 6.6a Mar 19 2010 Linux 2.6.32-23-generic
# //
# //  Copyright 1991-2010 Mentor Graphics Corporation
# //            All Rights Reserved.
# //
# //  THIS WORK CONTAINS TRADE SECRET AND
# //  PROPRIETARY INFORMATION WHICH IS THE PROPERTY
# //  OF MENTOR GRAPHICS CORPORATION OR ITS LICENSORS
# //  AND IS SUBJECT TO LICENSE TERMS.
# //
# do script_modelsim.val
# 1
```

Status: You can see the results here or in the file Simu_add_ex/SimuRes.txt

*Figure 12 : GANTT Chart of the synthesized design*

- **To view the simulation, click on the "Modelsim" button available at the bottom of the tool. This part will automatically generate the testbench for  verifying the design.**
- **Verify your simulations using the generated gantt chart and correlate them.**

** if you face some  error due to permission,  please inform the lab assistance during the lab session*

# TASKS FOR THE LABS

## TASK 1

### Objective

In the previous section you learnt about the basic usage of G.A.U.T. and synthesized an example C code. In this section you'll be asked to perform synthesis on different designs while varying various constraints.

### Set of Tasks

- **Re-synthesize this design by setting the cadency constraint to 140 ns and clock period to 10 ns.**

Question 06 : How many adders are generated this time?

____________________________________________________________

____________________________________________________________

Question 07 : Read the synthesis output of the tool and report the number of clock steps ? Is this information correct ? If not then how many clock steps should have taken to the tool to implement this design by using the number of adders reported in the previous question?

____________________________________________________________

____________________________________________________________

- **View the gantt charts "adder_1_UT" and "adder_1_UT_PIPE" generated by the tool.**

Question 08 : What is the difference between the two charts

____________________________________________________________

____________________________________________________________

- **Simulate your design and verify the results. This design would produce output after every N cycles*.**

  ** if you face some error due to permission, please inform the lab assistance during the lab session.*

# TASK 2

## Objective

In these set of tasks you will view the effects of varying the cadency constraint on the total gate count of circuit.

## Set of Tasks

- **Re-synthesize the same design by setting the cadency constraints to 190 ns, 140 ns, 90 ns, 50 ns and 30 ns.**
- **Fill Table 1 accordingly.**

Apply your previous knowledge gained in Digital Designs to calculate the gate count of the design. You might have to view the generated VHDL and the library viewer for reading the technology file.

| Cadency (ns) | Gate Count |
|---|---|
| 190 | |
| 140 | |
| 90 | |
| 50 | |
| 30 | |

***Table 01 : Gate Count of the Design***

Cadency value inferior to the latency value generates pipeline architectures.

## TASK 3

### Objective

In these set of tasks we will try to elaborate the relationship between latency and area of the design.

### Set of Tasks

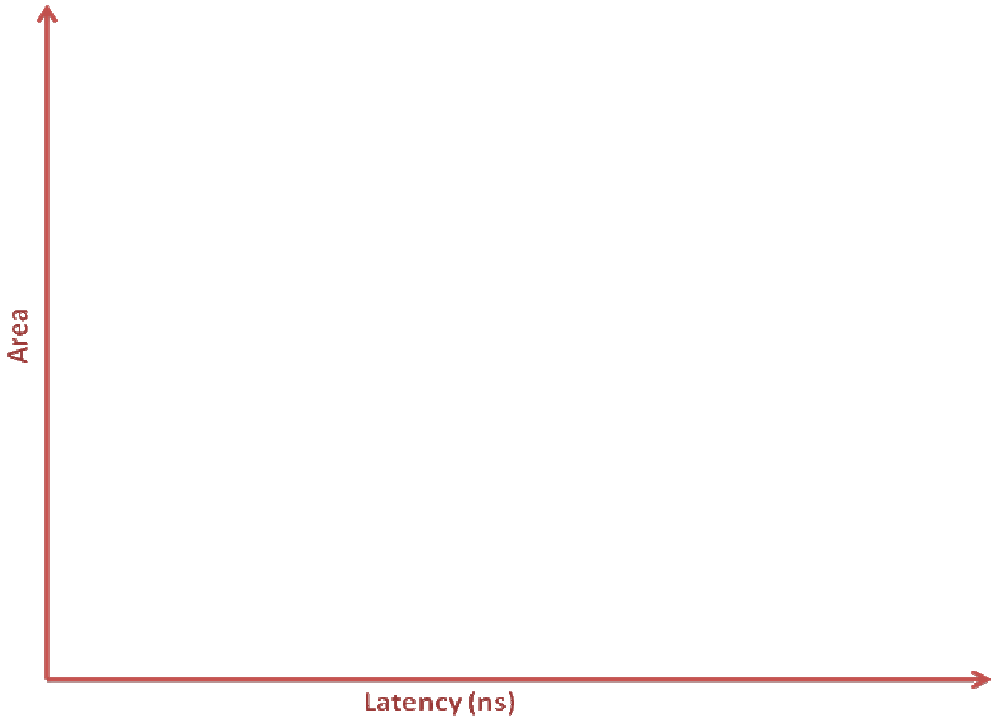
- **From G.A.U.T. main window select “C/C++ Compiler”.**
- **Click on the “Open” button and open “adder_2.c” by following the path :** $HOME/ASIC/Lab1/adder_2/adder_2.c”.
- **“Compile” the code and view the CDFG using the “Graph Tab”.**

Question 09 : Is this is a serial or parallel solution? How many adders are used by default?

__________________________________________________________

__________________________________________________________

- **Synthesize the design by setting the cadency value as 190ns, 140ns, 110ns, 70ns, 30ns, 10ns. Keep rest of settings as default and draw the graph represented below.**



## TASK 4

- **Set the cadency value as 110 ns, 50 ns and synthesize the design by using the default settings.**
- **Save the gantt charts by clicking at** 🖨
- **Use the following naming conventions :**

  **“adder_110ns.html” and “adder_50ns.html”.**

## TASK 5

### Objective

In this phase we will change the scheduling constraints on the same design and observe their effects on the synthesis process.

### Set of Tasks

- **Synthesize the design by setting the cadency value as 50 ns and scheduling strategy as "no_pipeline".**
- **View the synthesis report and respective gantt chart.**

Question 10 : How many resources are used and what is the latency of the design? Explain the effects of this scheduling step as compared to the TASK 4?

- **Save the gantt chart as "adder_50ns_nopipe.html".**
- **Synthesize the design by setting the cadency value as 50 ns and scheduling strategy as "no_pipeline". Also select "operator optimization".**
- **View the synthesis report and also save the gantt chart in "adder_50ns_nopipe_opt.html".**

Question 11 : Can you explain the effects of operator optimization? What happened as compared to previous design?

Question 12 : Compare the RTLs generated in Task 5. How they are different from each other

____________________________________________________________

____________________________________________________________

_____________________________________________________________

____________________________________________________________

## TASK 6

### Objective

In this phase we will change allocation constraint on the same design and observe their effects on the synthesis process.

### Set of Tasks

- **Set the cadency value as 50 ns and scheduling strategy as "no_pipeline". Also select "operator optimization". Select Allocation strategy as "manual".**
- **After clicking the "control" button, set 4 as number of adders.**
- **View the synthesis report and also save the gantt chart in "adder_50ns_manual.html".**

Question 13 : What happened? How results differ from the previous step ?

__________________________________________________________

__________________________________________________________

__________________________________________________________

__________________________________________________________

- **Change register allocation as "none".**

Question 14 : Compare your gantt chart with the previous one and explain what happened by using this allocation strategy.

__________________________________________________________

__________________________________________________________

__________________________________________________________

__________________________________________________________

## TASK 7

- **Complete the Table 2 based on task 3 to task 6. Use the information available in the gantt charts saved in these steps.**

| **Cadency Value** | **Operator Optimization** | **Scheduling Strategy** | **Allocation Strategy** | **Resources** | | **Latency** |
|---|---|---|---|---|---|---|
| | | | | **Registers** | **Adders** | |
| 110 ns | NO | Default | Automatic | | | |
| 50 ns | NO | Default | Automatic | | | |
| 50 ns | NO | No_pipeline | Automatic | | | |
| 50 ns | YES | No_pipeline | Automatic | | | |
| 50 ns | YES | No_pipeline | Manual | | 5 | |
| 50 ns | YES | No_pipeline | Manual | | 6 | |

## TASK 8

### Objective

We will apply different constraints on a FIR Filter which is a more complex design as compared to two input adders. We will apply different set of constraints and explore the design space in terms of serial/parallel tradeoffs.

### Set of Tasks

- **From G.A.U.T. main window select "C/C++ Compiler".**
- **Click on the "Open" button and open "fir.c" by following the path :** $HOME/ASIC/Lab1/FIR/fir.c".

Question 15 : What type of filter is this ?

____________________________________________________________

____________________________________________________________

- **Shows the impacts of latency on the area using the methods explained in previous tasks and complete the following table.**

| **Cadency** | **Allocation Strategy** | **Area** | **Multipliers** | **Adders** | **Buses** | **Latency** |
|---|---|---|---|---|---|---|
| 250 | global_ub | | | | | |
| 180 | global_ub | | | | | |
| 100 | global_ub | | | | | |
| 60 | global_ub | | | | | |

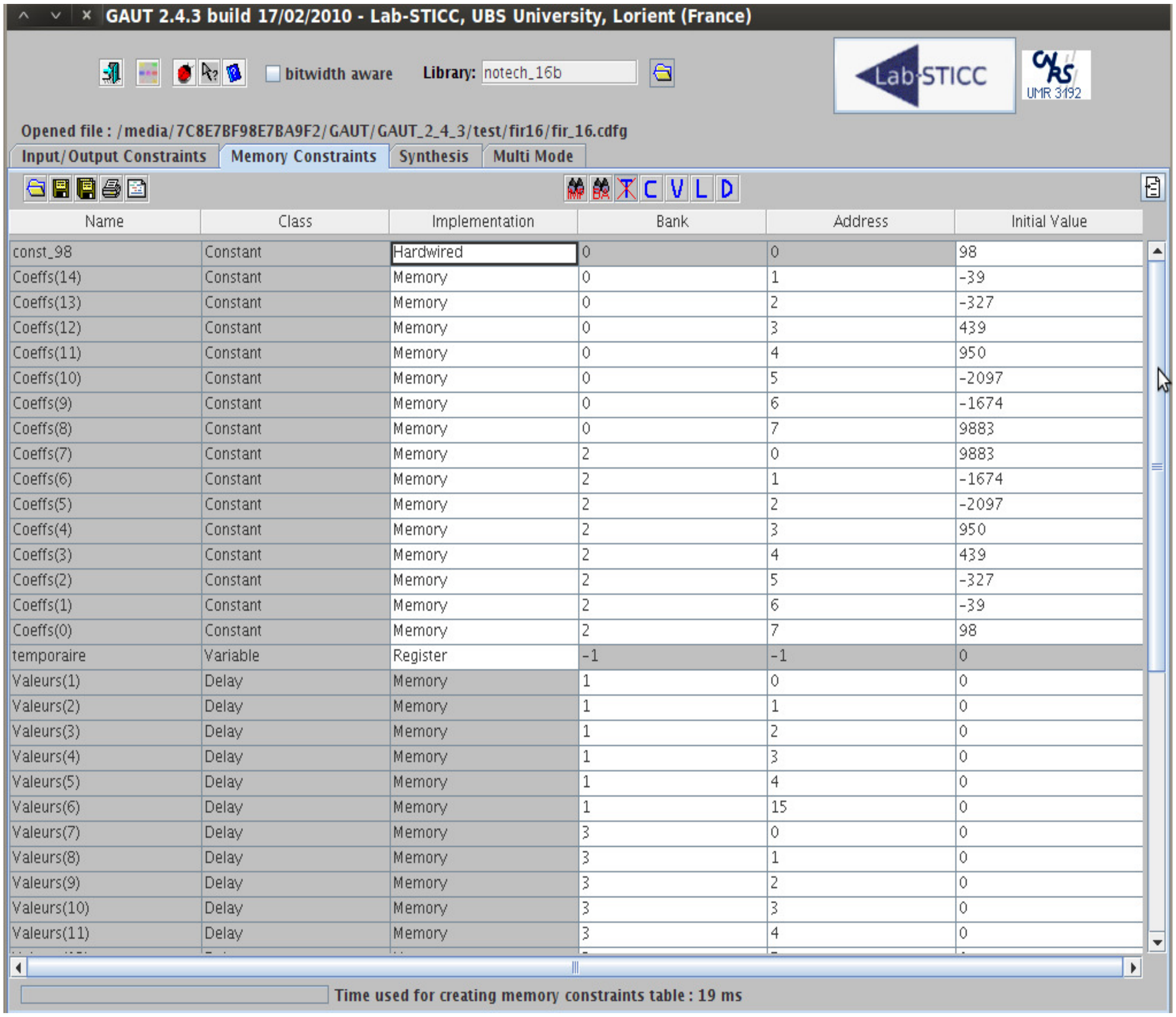
## TASK 9 (Optional)

### Objective

We will elaborate some more features of G.A.U.T. for controlling the HLS options.

### Set of Tasks

- **Click on the "Memory Constraints" tab in the VHDL synthesis mode and open the file "fir_16.cdfg"**



GAUT 2.4.3 build 17/02/2010 - Lab-STICC, UBS University, Lorient (France)

□ bitwidth aware    Library: notech_16b

Opened file : /media/7C8E7BF98E7BA9F2/GAUT/GAUT_2_4_3/test/fir16/fir_16.cdfg

| Name | Class | Implementation | Bank | Address | Initial Value |
|---|---|---|---|---|---|
| const_98 | Constant | Hardwired | 0 | 0 | 98 |
| Coeffs(14) | Constant | Memory | 0 | 1 | -39 |
| Coeffs(13) | Constant | Memory | 0 | 2 | -327 |
| Coeffs(12) | Constant | Memory | 0 | 3 | 439 |
| Coeffs(11) | Constant | Memory | 0 | 4 | 950 |
| Coeffs(10) | Constant | Memory | 0 | 5 | -2097 |
| Coeffs(9) | Constant | Memory | 0 | 6 | -1674 |
| Coeffs(8) | Constant | Memory | 0 | 7 | 9883 |
| Coeffs(7) | Constant | Memory | 2 | 0 | 9883 |
| Coeffs(6) | Constant | Memory | 2 | 1 | -1674 |
| Coeffs(5) | Constant | Memory | 2 | 2 | -2097 |
| Coeffs(4) | Constant | Memory | 2 | 3 | 950 |
| Coeffs(3) | Constant | Memory | 2 | 4 | 439 |
| Coeffs(2) | Constant | Memory | 2 | 5 | -327 |
| Coeffs(1) | Constant | Memory | 2 | 6 | -39 |
| Coeffs(0) | Constant | Memory | 2 | 7 | 98 |
| temporaire | Variable | Register | -1 | -1 | 0 |
| Valeurs(1) | Delay | Memory | 1 | 0 | 0 |
| Valeurs(2) | Delay | Memory | 1 | 1 | 0 |
| Valeurs(3) | Delay | Memory | 1 | 2 | 0 |
| Valeurs(4) | Delay | Memory | 1 | 3 | 0 |
| Valeurs(5) | Delay | Memory | 1 | 4 | 0 |
| Valeurs(6) | Delay | Memory | 1 | 15 | 0 |
| Valeurs(7) | Delay | Memory | 3 | 0 | 0 |
| Valeurs(8) | Delay | Memory | 3 | 1 | 0 |
| Valeurs(9) | Delay | Memory | 3 | 2 | 0 |
| Valeurs(10) | Delay | Memory | 3 | 3 | 0 |
| Valeurs(11) | Delay | Memory | 3 | 4 | 0 |

Time used for creating memory constraints table : 19 ms

*Figure 13 : Synthesis with Memory Constraints*

This table allows to assign variables and/or constraints in different memory banks. The basic strategy consists in placing those operands at different memory banks, which are presented together to the operator. Grey blocks cannot be modified while white fields can be modified by entering/selecting the desired values.

- **goto your FIR directory and open the file fir.mem.** This file shows the I/O chronogram and the temporal access conflicts. The contents of this file inform us that memory data_in and coeff (7) cannot be in the same bank, X[1] and coeff[6] cannot be in the same bank. A simple solution is to place coeffs and samples in different memory bank.
- **Fill in the following table accordingly after editing the two lines of fir.c as :**

```
#define N 16
static const int Coeffs [N] = {98,-39,-327,439,950,-2097,-1674,9883,9883,-1674,-2097,950,439,-327,-39,98};
```

| | **Synthesis Options** | | | | **Results** | | | |
|---|---|---|---|---|---|---|---|---|
| | Cadency (ns) | MC | Num. of MB | CL | Num. Buses | Num. Adders | Num. Mul | Num. Regs |
| Synthesis 1 | 200 | No | - | - | | | | |
| Synthesis 2 | 100 | No | - | - | | | | |
| Synthesis 3 | 100 | Yes | 2 Banks | Memory | | | | |
| Synthesis 4 | 100 | Yes | 4 Banks | Memory | | | | |
| Synthesis 5 | 100 | Yes | 2 Banks | Hardwired | | | | |

## 5. References

1. G.A.U.T. user Manual
2. How to write a C Code in G.A.U.T.